THE BCS PROFESSIONAL EXAMINATION
Diploma

April 2005

EXAMINERS' REPORT

Software Development Environments

Please note: this module will no longer be examined.

General
It is necessary for the examiner to be able to read your answer. Try to write legibly.

Question 1
This question examines section 1 of the syllabus, "The Software Development Lifecycle"

1. *a)* Write down EIGHT components of the software development lifecycle and then explain why the production of software is not just an eight stage linear process but instead the word 'cycle' is used. **(10 marks)**

b) Explain, with examples, why the process of describing what a program is to do, designing the program and writing it are three separate activities. **(15 marks)**

Answer Pointers
Part (a)
[see syllabus] Requirements, Prototyping, Specification, Design, Coding, Implementation & Testing, Documentation, Maintenance *(4 Marks)*

At any stage during the initial construction of a piece of software from Prototyping onwards it may be that something is discovered that requires returning to some earlier stage. This can cause 'mini' cycles. However the overall concept is that after a piece of software has been put into use, it may be that an alteration or extension is required which means that the Requirements have altered and so the whole sequence of stages is revisited. *(6 Marks)*

Examiner's Comments
Generally answered satisfactorily.

Part (b)
Looking for a clear separation of specification, design and implementation using examples.
- specification describes what answer is required,
- design describes an algorithm to calculate the answer and
- implementation is an encoding of the algorithm using specific computing data structures and program structure.

(5 marks for each of the three descriptions each accompanied by an example)

Examiner's Comments
Few candidates managed to include any examples although the question clearly asks for them.

Question 2
This question examines section 2 of the syllabus,, " The Programming Environment"

2. *a)* "A beginner writes a whole program in a single file and runs it using an interpreter. A professional constructs a project and uses a compiler." Referring to this scenario, explain the difference in scale of the programs involved and the different demands made on the language implementation system used.
(10 marks)

b) Under one operating system a program is supplied with data by dragging a data file onto the program icon. In another the name of the program is typed followed by the name of the data file. Contrast the two approaches and describe scenarios where each is shown at its best. **(9 marks)**

c) What precautions should be taken when writing an application in a high level language to ensure cross-platform compatibility? **(6 marks)**

Answer Pointers
Part (a)

Single file, interpreted:
- Simple
- Fast 'translation' time
- slow 'run' time
- suits learner

Project, compiled
- Sophisticated
- multiple source documents
- multiple object documents (some from system libraries, some from programmer)
- Slow 'translation' time
- fast run-time
- suits production program

(1 mark for each bullet point making 10 marks in total)

Examiner's Comments
Generally well answered.

Part (b)
Dragging data onto program icon is
- the graphical, windows oriented style.
- It is very easy to learn
- and perform
- and places no burden on the memory.

Typing a command and data is
- harder to learn,
- harder to perform
- and requires the user to remember the commands.

Drag & drop is well-suited
- to individual program executions because it is quick and easy interaction for the user

The command line structure is superior
- for a repetitive task like "process all the data files in the directory that have filenames like x?y*", using wildcard matching and/or script files

(1 mark for each bullet point making 9 marks in total)

Examiner's Comments
Generally well answered, except for the identification of tasks for which each was superior.

Part (c)
Portability

- Use only the standard language (no extensions)
- Try not to rely on potentially machine specific details (e.g. word length, range of integers, etc)
- Isolate likely problem areas (e.g. graphical interface) to as small a section of the program as possible

(2 marks for each bullet point or similar relevant precautions making 6 marks in total)

Examiner's Comments
This part was left blank or poorly answered by most candidates.

Question 3
This question examines section 3 of the syllabus "Program Design"

3. *a)* When a program is executing, data can be read from a file and results can be written to a file. Describe and compare the following terms relating to files:
i) Text file/binary file
ii) Sequential/direct access
iii) Local/remote
iv) Open for reading/writing/appending **(18 marks)**

b) In most programming languages there is a "file open" and a "file close" command. Describe the purpose of these two commands and include information on the parameters which are used and any results obtained. **(7 marks)**

Answer Pointers
Part (a)

- **A Text** file is a sequence of bytes representing ASCII characters
- **A Binary** file is also a sequence of bytes but the interpretation that is put upon them is not fixed. Groups of bytes might represent integer or real numbers or ASCII strings depending on the application
- **Sequential access** is where the reading position moves forward only through the file allowing the bytes to be read in order, once only.
- **Direct access** allows the reading pointer to be placed at any point in the file at any time
- A **local file** is in the file system of the computer running the program.
- A **remote file** is accessed across a network of some kind
- **Open for reading** means the file cannot be changed.
- **Open for writing** means arbitrary change.
- **Open for appending** means only allowed to add bytes at the end.

(2 marks for each bullet point making 18 marks in total)

Examiner's Comments
Generally well answered.

Part (b)
File open usually has 2 parameters

- The name of the file to be opened
- The style of opening (r-ead, w-rite, a-ppend)

Return

- A reference to the file is normally returned as a result

The open routine has to

- find the file, create a buffer,
- signal an error if file missing, mark the file as 'in use', etc.

File close usually has one parameter

- The reference to the file to be closed

Close action

- For a file being written or appended the close command has to empty any remaining data from the buffer to the file. The file should now be marked free to be used by other programs.

(1 marks for each bullet point making 7 marks in total)

Examiner's Comments
Candidates did not seem to have had the experience of using these facilities from within a programming language as there were few confident answers - some merely described File|Open, File|Close as provided on menus in WIMP apps]

Question 4

4. *a)* Identify a program development environment with which you are familiar and discuss THREE features it offers to assist individual programmers to develop code. **(10 marks)**

b) Describe THREE features of a debugger that can be used to assist in the identification of errors within a computer program. **(9 marks)**

c) Identify THREE compiler options with which you are familiar and indicate the circumstances under which each of these would be used. **(6 marks)**

Answer Pointers
Part (a)
The identification of any suitable environment, e.g. MS Visual Basic (Many candidates failed to state which environment they were discussing)

Example features include (any three acceptable):

Auto completion of code
For example, when a programmer types the first few characters of a keyword, the environment completes the rest

Automatic indentation
Body of functions, loops etc indented according to predefined rules.

Colour Coding
Highlights keywords, comments etc and helps identify typographical errors such as spelling or missing closing comment token.

Syntax Directed Editing
Automatic insertion of closing tokens when opening token entered (e.g. BEGIN/END combinations).

Context Sensitive Language help
Provides help on the keyword/function under the cursor, e.g. indicating the nature of parameters required for a function.

(1 mark for each feature identified, 2 for associated description up to a maximum of 9 marks)

Part (b)
In general, this part was well answered.

Examples include: (1 mark for feature, 2 for description)

Break points
Allow the program to be run at normal execution speed until the textual area of the program where an error is thought to be located is reached. Thus breakpoints speed up the debugging process.

Watches (also known as variable inspectors)
Allow the user to view the value of an individual variable and/or an expression at any desired stage in the execution of the program. Watches can be used to view the values of internal variables at predetermined points (at which expected values of the variable can be pre-determined) and hence narrow down the location of an error.

“Step over” and “Step into” operations
These commands allow the code to be executed statement by statement. In particular, when executing function calls, the “step over” command treats the function call as a single statement, ignoring the internal operation of the function. In contrast, the “step into” command causes each individual statement within the function to be executed when determined by the user (using this “stepping” functionality).

Part (c)
Several candidates simply did not attempt this question, and of those that did, many presented irrelevant material.

Any three compiler options were acceptable. The following are examples:

Define target platform
Used when the same compiler can generate code for several different hardware platforms and the software is to be deployed in a multi-platform environment.

Optimise code
Typically code can be optimised for either speed or size. If the software is for an embedded system, the size of the executable may be more important than execution speed. However, if the code involves millions of complex calculations speed optimisation may be preferable.

Include/Exclude debugging code
Typically developers would include debug code when building for test purposes and exclude his code for a final shipping version, increasing the efficiency of the final software product.

Question 5
This question examines Section 5 of the syllabus, “Program Testing.”

5. *a)* Describe TWO *software tools* (i.e. not ‘processes’ or ‘methods’ such as top-down or bottom-up) that can be used to assist the testing process. Your answer should highlight the benefits of using each tool, give an example of where it would be appropriate to use it and identify any issues with which a user of the tool should be familiar. **(10 marks)**

b) Testing is often divided into unit testing, module testing, sub-system testing, system testing and acceptance testing. Explain the role of each of these individual stages, identifying the types of error likely to be detected at each stage. **(15 marks)**

Answer Pointers
Part (a)
In general, this part was well answered.

Any class of testing tool is acceptable, but examples (to indicate the level of detail required) are:

Test Data Generator
Description
- Generates a large number of test cases using a specification of the syntax of the input.

Benefits
- Automates the generation of test cases, thus saving time (and therefore money)

Typical Uses
- Testing performance in a practical environment, e.g. a database system dealing with a large volume of transactions.

Issues
- Do not relieve the burden of checking test outputs.
- Need to ensure syntax of test inputs is accurate.

Simulator
Description
- Simulates a real-world environment. Often used in the testing process where bugs could be catastrophic.

Benefits
- Allows software to be tested in a “safe” environment without potential damage to expensive hardware or other similar risks

Typical Uses
- Testing software for a nuclear reactor (or similar) or where it integrates with expensive hardware and bugs could damage this.

Issues
- There could be bugs in the simulator.
- Not always possible to simulate the real world accurately.

Part (b)
Some candidates did not refer to the stages identified, but rather split the stages into other groupings with which they were familiar.

Unit Testing
- Individual functions are testing in isolation.
- Typically logical errors within the function being tested are detected at this stage.

Module Testing
- Collections of dependent components (such as an object) are tested.
- Typically logical errors within the flow of control mechanisms of the module are detected at this stage.

Sub-system Testing
- Involves the testing of collections of modules which together form major components of the overall system.
- Typically logical errors within the sub-system flow of control mechanisms are detected at this stage along with interface errors such as the incorrect number of parameters for a function call.

System Testing
- Here the complete system is tested as a whole.
- Typically errors in the interface between sub-systems are detected along with logical errors in the system's overall flow of control code.

Acceptance Testing
- Here the system is tested with data from the system procurer.
- Typically errors or omissions in the system requirements are detected at this stage along with performance related problems.

Question 6

This question examines Section 6 of the syllabus, "Quality Assurance and Documentation."

6. *a)* Identify FIVE measures of software quality. For each, explain what it attempts to measure and outline how "high quality" software (as indicated by the measure) can be achieved. **(15 marks)**

b) To remain useful, most commercial software is regularly updated after the initial installation phase. Describe how the process of deciding which requests for changes should be implemented could be managed within an organisation that receives many requests for such updates. **(10 marks)**

Answer Pointers
Part (a)
Many candidates failed to identify how "high quality" software, as defined by the specific metric, could be achieved.

Examples of acceptable answers:

Re-usability
- Refers to the easy with which the code could be reused within similar applications
- Can be achieved through the use of modularisation in which modules are loosely coupled but highly cohesive

Portability
- Refers to the ease with which the code can be transferred to other hardware platforms.
- Can be achieved by isolating hardware dependant code from the main application logic

Maintainability

- Refers to the ease with which programs can be updated to extend their useful life.
- Can be achieved through good programming practice, e.g. commenting, use of meaningful variable names, modularisation etc. Can also choose the simplest algorithm where alternatives exist.

Reliability

- Refers to the probability that the program will operate as intended.
- Can be achieved through adequate testing, including appropriate load testing.

Clarity

- Refers to the ease with which the code can be read and understood.
- Can be achieved by using the same techniques as maintainability.

Part (b)

Typically solutions to this part were very brief and lacked detail. It was not sufficient to write 5 lines to gain the full 10 marks. The solution below is indicative of the sort of answer required but is not the only possible one.

Requests for changes should be submitted on a standard form to include the following information:

- Name and contact details of person making the request
- Date of request submission
- A checkbox indicating whether the request relates to an error in the existing software or a request for additional functionality
- A description of the requested change

Bug fixes go to the support/maintenance team

- A check is made to see if the bug has previously been reported. If so, the additional report form is filed with the others already received.
- New bugs are allocated a priority level (say 1-5) depending on (for example) the severity of the problem, the number of customers impacted, the time required to fix the fault etc. If an additional forms are received reporting the same bug, the priority may be changed.

Requests for new functionality go to the development team

- Again a check is made to see if the same or similar request has been made previously and if so, the form filed with the previous ones.
- Again each form is allocated a priority level depending on (for example) the number of customers likely to benefit from the new functionality, the amount of time required to implement the new feature. Additional forms requesting the same change may alter the priority assigned.

Forms are filed based on the module or area of the program’s functionality that is impacted.

Issues with the highest priority are implemented first.

However, need to ensure that request with low priority are not permanently ignored (particularly bug fixes) and hence the length of time in the “queue” may increase the priority of a request.